UiO : Department of Informatics
University of Oslo

A Population-Based Incremental Learning Approach to Network Hardening

Constrained optimization network hardening

Alexander Paulsen

Master's Thesis Spring 2015

A Population-Based Incremental Learning Approach to Network Hardening

Alexander Paulsen

May 18, 2015

Abstract

Enterprise networks constantly face new security challenges. Obtaining complete network security is almost impossible, especially when usability requirements are taken into account. Previous research have provided ways to identify attack paths due to network vulnerabilities and misconfiguration, but few have addressed ways to correct them, especially when considering usability requirements. This thesis presents an approach based on the learning algorithm Population Based Incremental Learning in order to solve a constrained optimization problem with the intention of increasing network security. Preliminary results show that this approach is effective, scalable and reliable.

Acknowledgements

For his support, valuable input and contribution of exceptional ideas, I would like to express a special gratitude to my supervisor Anis Yazidi. His influence and passion for this field of study has inspired me enormously. A special thanks to Boning Feng for co-supervising me and providing valuable suggestions throughout my research.

Thanks to Xinming Ou for his positive feedback and his tool MulVAL which has made this research possible. A gratitude to Su Zhang, for providing MulVAL input-files for my experiments.

From a more personal perspective, i would like to extend an inexpressible gratitude to my family. I am heavily indebted for their support and sacrifice.

Contents

List of Figures

List of Tables

Part I

Introduction and Background

Chapter 1

Introduction

This is a master thesis written as a part of the masters programme *Network & System Administration* at the University of Oslo in collaboration with Oslo & Akershus University College of Applied Sciences. The purpose of this thesis is to understand computer security as a combination of several vulnerabilities in a network and how they are linked together. The result of this study will be to suggest a solution in how to improve network security. Having that said, network security should not be at the expense of usability requirements such as requirements of availability of services and resources. Network security is a complex task of having a balance between security and usability where both ends must compromise in order to accomplish such a balance.

In the past years, tools and techniques associated with targeted attacks have become easier to obtain and utilize. Nation states, private companies and criminal groups have demonstrated desire to use such tools to execute high-profile attacks with the intention of destructive or information-stealing purposes such as seeking development and research data, financial information and intellectual property. A recent report ("Security Report 2015," 2015) states that the number of targeted attacks have increased, but the awareness of such attacks has not.

1.1 Network security

Networks, especially within enterprises, are growing in complexity and size. Network security is a large topic that essentially addresses how to secure a network infrastructure. The attention towards IT security has increased dramatically in the past years and it has revealed a tremendous amount of security breaches. Motivations behind such breaches are of financial gain driven by groups or individuals, industrial or governmental espionage, theft and damages, or just out of thrill. This is a result of the expanding reach of the Internet and increased availability to services

and information channels which businesses and organizations have moved online.

Security personnel or system administrators are in a constant race to keep up with new threats on a daily basis. There is a variety of tools and techniques available to strengthen network security such as vulnerability scanners and attack graphs. Other examples include firewalls, intrusion detection and prevention systems, advanced malware detection, anti-virus software, access control, log analysis systems and VPN-solutions. What they are all intended to is to prevent or detect unwanted activities such as malicious intrusions.

1.1.1 Malicious network intrusions

Malicious intrusions into computer networks are critical to enterprises making protection crucial to ensure stability. It is normal to assume that confidential and sensitive data is stored within the network of an enterprise. Moreover, a network of functioning computers keeps it running on a daily basis. A malicious intrusion can potentially paralyze the enterprise's activities, damage its products and economy, ruin its reputation and destroy its credibility. In contrast to materialistic losses, it is difficult to determine the total losses as a consequence of such an intrusion.

Targeted attacks usually exploit multiple vulnerabilities in a so-called multi-stage attack to elevate their privileges and access in a network, hopefully evading security mechanisms. Intrusion detection systems generate large amounts of alerts, and for a system administrator, it is easy to overlook an ongoing multi-stage attack.

1.1.2 Securing a network

System administrators need to secure their network to keep their resources safe from malicious activities. There are several steps towards securing a large network, and one of the most crucial is to secure it against known identifiable threats. Many enterprises spend effort into doing so and a result of this would hopefully be to have more secure systems and thus prevent a compromised network. A vulnerability analysis implies that every vulnerability will be removed, but in practice, this is usually not the case. Removing vulnerabilities is complicated through reduced availability of patches and upgrades, cost, and demands regarding efficiency, usability and uptime.

Being able to recognize system vulnerabilities is considered to improve the awareness of the network security. Vulnerability scanners are tools which can assist a system administrator in identifying vulnerabilities on a system (Holm, Sommestad, Almroth, & Persson, 2011). However, these scanners

can only identify vulnerabilities in isolation. Attack graph tools can enhance the information from a vulnerability scan to give a more contextual meaning in a graphical way. Vulnerability databases are also useful as they provide enhanced information about every known vulnerability.

1.2 Thesis statement

Previous work have given solutions in how to identify network vulnerabilities, but few have addressed how to correct them. Vulnerability scanning of large networks result in massive amounts of data, which can be used to generate a complex attack graphs, often unreadable for humans. One of the intentions of the attack graph is to give a basis for future security decisions. However, graphs which are not human-readable cannot provide efficient and satisfactory basis for decisions. There are few available methods in how to efficiently address all of the vulnerabilities available in a network and use the information to improve network security in addition to taking usability and network requirements into account. This leads us to the following question:

Q: How can we use attack graphs to efficiently secure a network while maintaining usability?

1.3 Thesis objectives

As an attempt to answer the problem statement, the target of this thesis is to find a way to organize the information from an attack graph in order to systematically and efficiently find an improved and more secure solution to an already vulnerable network. The solution should take usability and other requirements into account. Using a suitable algorithm to calculate such a solution is required to ensure efficiency and reliability.

1.4 Thesis outline

This thesis is organized into the following chapters.

Chapter 1. Introduction:

This chapter provides a general information on the topic, which leads into what point I want to do in this thesis. This is outlined in the *thesis statement* in this chapter.

Chapter 2. Background:

The background chapter introduces system vulnerabilities and vulnerability scanners followed by a description of attack graphs and the existing

attack graph tools. As the thesis will utilize the attack graphing tool MULVAL, a more detailed elaboration on this tool is carried out. Furthermore, network hardening is described and previous work related to the problem statement is outlined. Finally, an introduction to Learning Automata is portrayed and a detailed description of the algorithm Population Based Incremental Learning.

Chapter 3. Approach:

The chapter describes the solution and how it should be implemented. Examples show what is expected of the solution. It also states the various experiment environments.

Chapter 4. Results:

Results from the experiments are presented here.

Chapter 5. Analysis:

This chapter interprets the results and states whether the presented solution is reliable.

Chapter 6. Discussion and conclusion:

Reviews what has been accomplished in this thesis, gives suggestions for further work and issues which occurred during the project. Lastly, the chapter links together the whole thesis, mostly focusing on the analysis and discussion which are compared to the thesis statement.

Parts:

These chapters are structured in three parts. Part 1 consist of chapter 1 and 2, part 2 has chapter 3, and part 3 contains chapter 4 through 7.

Chapter 2

Background

Computer systems play a critical role in most parts of our society. Critical information and infrastructure such as financial data, power grids, governmental and enterprise secrets and more are controlled by these systems. Protecting these assets is crucial to our society. In order to fully comprehend the context of this thesis, an introduction to different technologies and expressions is needed.

2.1 System Vulnerabilities

Software vulnerabilities are usually found in most software available. Intrusions are commonly performed by exploiting a system due to vulnerabilities and misconfiguration in order to gain increased privileges. System vulnerabilities are often discovered by scrutinizing a system, normally performed by security professionals or hackers with malicious or economic intentions. Some vulnerabilities are even discovered by chance. Vulnerabilities are discovered in a tremendous variety of applications, making even a medium sized network vulnerable on many levels. This could potentially mean that one vulnerability gives privileges to exploit another.

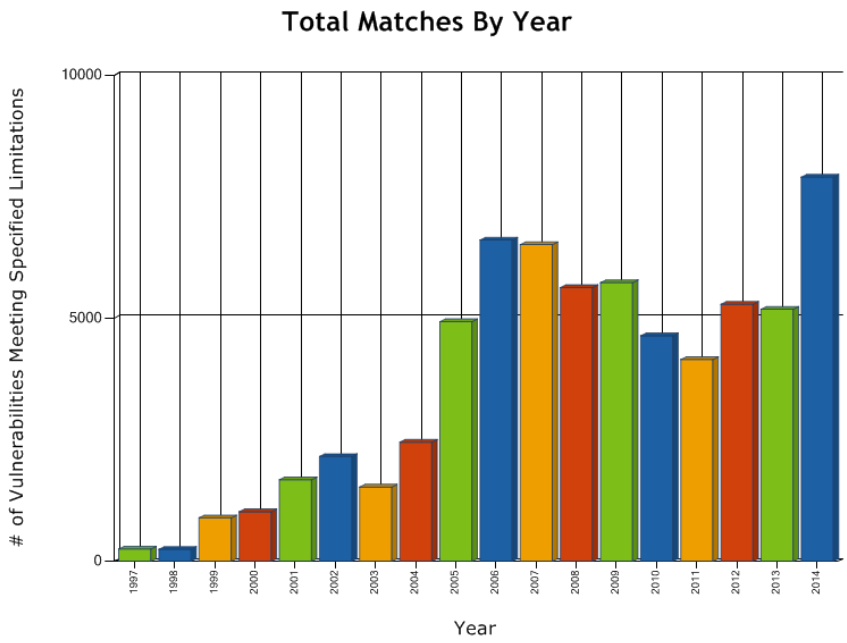

Figure 2.1: Yearly discovered software flaws (CVE) since 1997 (NVD, 2015).

There are around 70000 Common Vulnerabilities & Exposures (CVE) known to date according to the National Vulnerability Database (NVD) (NVD, 2015). It is a daunting task to manually keep track on all existing vulnerabilities and detect those available in a network. Fortunately, there are tools available to automate these processes called network vulnerability scanners. Examples include Nessus, OVAL, Core Impact and more. These scanners are used to examine the architecture of a network and report identified vulnerabilities. A typical network vulnerability assessment is commonly divided into three parts: network scanning, vulnerability scanning and vulnerability analysis (Holm et al., 2011).

Network scanning identifies which hosts are available in a network, what operative systems they run and what type of services they are using. The next step involves the actual *vulnerability scanning*. A database of vulnerability signatures are compared to the information from the network scanning to generate a set of vulnerabilities that are supposedly available in the network. The next step is to verify the presence of the set of supposedly available vulnerabilities by actively trying to exploit the systems. This is often executed by sending specially crafted packages designed to exploit the vulnerability in question. An example of such an exploit would be to try exploiting a vulnerability in the Network Time Protocol (NTP) using an attack called the *NTP reflection attack*. This attack is performed by sending a crafted packet which requests a large amount of data from the server by taking advantage of a flaw in older versions of NTP (ICS-CERT, 2014). The flaw can be used to collect data of hosts connected to the NTP server, but is also greatly considered to serve as a Distributed

Denial of Service (DDoS) attack as a small query (the crafted packet) can redirect large amounts of data. The scanning tool will know whether a NTP server is vulnerable if it receives a reply with the queried data. From a scanning tool's point of view, it is essential that performing these active tests does not disrupt the services. Lastly, a *vulnerability analysis* evaluates the severity of all identified vulnerabilities. Most enterprises have a large amount of vulnerabilities, and it is therefore important to identify which vulnerabilities represent the highest security risks. Some vulnerabilities are easy to exploit or constitute huge consequences. It is thus essential to determine which vulnerabilities are most significant.

CVE	Description
CVE-2010-0425	A module in the web service Apache called *mod_isapi* can be forced to unload a library before a request has completed. It can be performed by sending a specially crafted request to the web server. The exploit can result in memory corruption, allowing an attacker to run arbitrary code on the server. This vulnerability is restricted to Apache running on Windows
CVE-2013-0130	CoreFTP is vulnerable to buffer overflow when parsing longer than usual directory names from a FTP server. The commands *LIST* and *VIEW* can result in denial of service meanwhile the command*DELE* can cause arbitrary code execution.
CVE-2014-0160	A vulnerability in *OpenSSL* cryptographic software library allows any attacker to retrieve memory data which under normal conditions are protected by SSL/TLS encryption. This CVE is commonly known as the Heartbleed bug and is exploited by sending a crafted request to a server revealing usernames, password and other content located in the memory.

Table 2.1: Examples and descriptions of vulnerabilities (CERT/CC, 2015).

However, most vulnerability scanners can only evaluate one vulnerability by itself and rank or score it accordingly. They are unable to see the entire picture of all vulnerabilities as a whole.

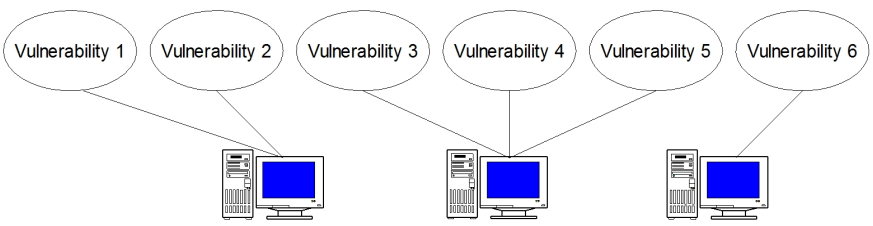

Figure 2.2: Vulnerability scanning.

Figure 2.2 shows that results from a vulnerability scan show what vulnerabilities each host has. This means that security predictions are limited to one single host at a time. In order to solve this problem, attack graphs have been proposed. They rely on the information from vulnerability scanners in order to provide enhanced vulnerability management.

2.2 Attack Graphs

For protecting a network against attacks, there are numerous vulnerability scanners, Intrusion Detection Systems (IDSs) and Advanced Malware Detection systems (AMD) such as Snort, Suricata, ISS, Tipping Point, FireEye, ElJefe, Nmap, Nessus, OVAL and Cisco Security Scanner. These solutions can be applied in real networks and can prevent future attacks. The previously mentioned technologies have been developed to identify threats and vulnerabilities and they have become extremely useful and powerful. However, they do not verify that all conditions for a complete attack are met, nor do they take into account that a set of multiple linked attacks are potentially more harmful than individual ones. In order to evaluate the security of a network of hosts, it is insufficient to consider isolated vulnerabilities on each host, which only provides a partial picture since sophisticated attackers perform advanced multi-stage attacks.

Phillips and Swiler (1998) propose an approach to graphically represent the relations of vulnerabilities in a network. The concept is called attack graph. An attack graph is a dedicated tool for network security risk assessment as it is tedious to manually deduce security risk in medium-size to large networks. It represents prior knowledge about network vulnerabilities, their dependencies and network connectivity. They present their own model which inputs network configuration and topology information, attacker profiles and a database of attacks. Phillips and Swiler state in their article that by assigning probability of success or cost as in the level of effort for the attacker, one can use various algorithms to identify paths with highest probability of success. This thesis is based upon this statement.

Attack graph research particularly focus on three different goals (Lippmann & Ingols, 2005). One of them is specifically directed towards network security analysis. These attack graphs often take into account that an attacker starts from a specific location in order to determine whether the attacker can gain normally restricted privileges on one or several targets. Another goal is to present a formal language used to describe states and actions by defining the needed preconditions necessary for an attack to take place and the post-conditions which represent the changes in a network state after a successful attack. A third approach focuses on how attack graphs can be used to organize large numbers of IDS alerts. If an IDS alert can be associated with an action in the graph, alerts can be grouped into

known stages for an attack path identified by the attack graph. This is useful for detecting the exact stages an attacker is performing in a successful attack. A mechanism for matching IDS alerts to stages in an attack path is required in this approach.

A typical procedure when analyzing network security is described in Figure 2.3 (Sheyner & Wing, 2004).

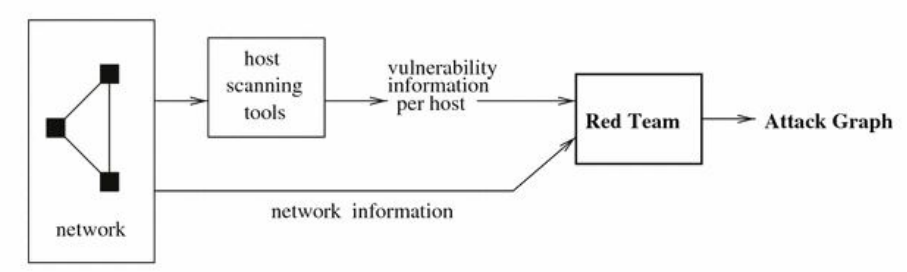

Figure 2.3: Network vulnerability analysis.

Initially, scanning tools determine available vulnerabilities on each host. Other network information, such as availability through connectivity between hosts, is used to produce an attack graph. An attack graph can show relations between existing configurations, exploits and undesired privileges between all hosts in a network. Figure 2.4 is an example of such a relation, limited to one exploit on one host. This example has PRE and POST conditions. The configuration-entities represent the preconditions, meanwhile the privilege-entity represents the postcondition.

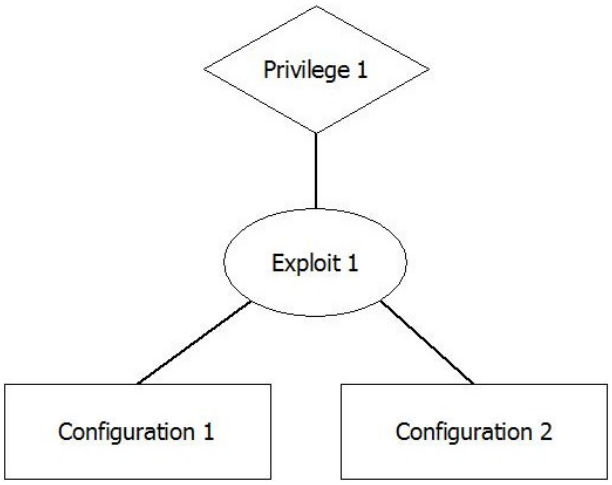

Figure 2.4: Example of simple attack graph showing relations between configurations, exploit and privilege.

One important aspect when it comes to attack graphs is that an exploit

can only be realized when *all* of its required conditions are satisfied. In Figure 2.4, the configuration entities (Configuration 1 and 2) represent the exploit conditions in the given example. We call these AND-conditions. A condition, however, can be satisfied if *any* of the realized exploits implies the condition. We call these OR-conditions. In Figure 2.5, only one exploit have to be realized in order for the privilege condition to be true. We say that the require relation is conjunctive, meanwhile the imply relation is disjunctive.

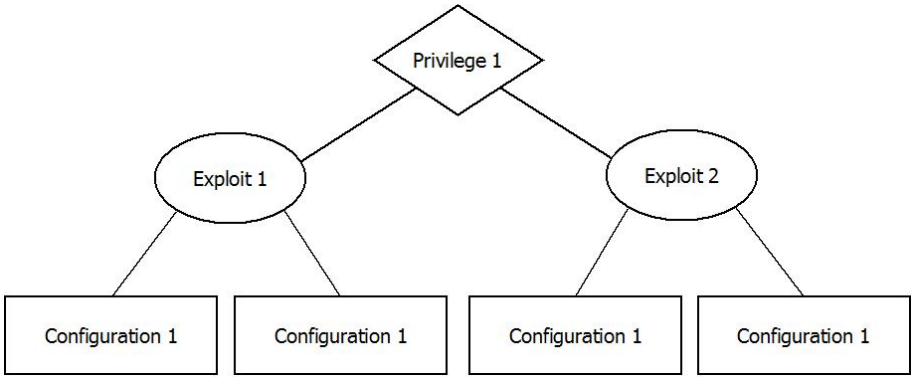

Figure 2.5: Example of simple attack graph showing relations between configurations, exploit and privilege.

Multi-stage attacks take place when an attacker can elevate his privileges multiple times by having the opportunity to exploit another vulnerability from the previously increased privilege. Figure 2.6 exemplifies this.

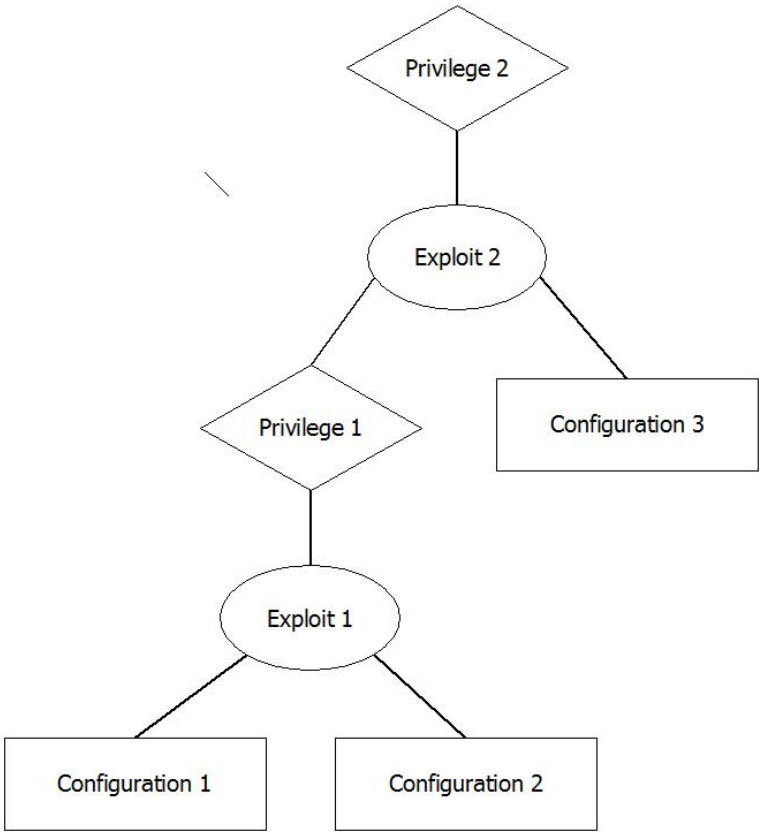

Figure 2.6: Example of simple attack graph showing relations between configurations, exploit and privilege.

In this example, an attacker can continue the attack into the network with his increased privileges he gained from exploiting *Exploit 1*. The objective for a multi-stage attack would be to:

1. Gain specific privileges to an unique host in the network.

2. Gain general privileges to multiple hosts in the network.

Any of the two outcomes are highly unwanted. Having that in mind, an attack graph can help answering the following questions:

- "What vulnerabilities exist on my systems?"

- "In how many ways can an attacker reach his goal?"

- "What actions can be taken or prevented to ensure the attacker does not achieve his goal?"

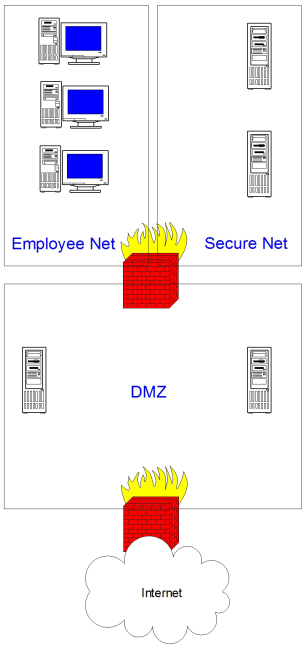

Figure 2.7: A typical network topology.

Figure 2.7 illustrates a simple network topology. An attacker coming from the Internet would initiate an attack by exploiting available nodes in the demilitarized zone (DMZ). A multi-stage attack does not stop there. Such an attack continues using the increased privileges on a node which has been exploited in the initial attack to exploit more nodes, preferably in a secured subnet, where the most important resources are located.

2.2.1 Tools for generating attack graphs

There are multiple attack graph tools available. Here some of them are listed:

- Topological Analysis of Network Attack Vulnerability (TVA) (Jajodia, Noel, & O'Berry, 2005) was developed at the George Mason University and is an example of such an attack graph tool. The idea is to use an exploit dependency graph to show pre and post conditions for all exploits. It relies on input information from Nessus scans. A graph search algorithm is used to chain the vulnerabilities in order find and represent different attack paths which include multiple vulnerabilities.
- Cauldron (O'Hare, Noel, & Prole, 2008) is a commercial version of TVA. The tool works in three steps: firstly import all information in a targeted network. Secondly, associate the information with the known vulnerabilities. Lastly, provide a modelling environment for analysis. Cauldron works with the vulnerability scanners Nessus,

FoundScan, Symantec Discovery and Retina. Initially, the tool did not scale well as the algorithm in the worst case was O(n^4) or O(n^6) where n is the number of hosts in the network (Yi et al., 2013). This has however been improved with O(n^2) as the worst case.

- Network Security Planning Architecture (NetSPA) was developed by Ingols and Lippmann (2006) in Lincoln Laboratory of MIT. The intention is to model adversaries and measure the effect of simple counter measures. It uses firewall rules and results from vulnerability scans to create a network model which is utilized to compute network reachability and the attack graph. The risk is assessed by measuring the total assets that can be controlled by an attacker. It is therefore useful for identifying the most valuable attack paths in a network and propose suggestions in how to repair the most serious weaknesses quickly.
- Multi-host, multi-stage Vulnerability Analysis (MulVAL), an open source project developed by Ou in collaboration with Govindavajhala and Appel (2005) at Princeton University. MulVAL uses information from vulnerability databases, configuration information from each node as well as other relevant information to graph the pre and post conditions from each exploit and how they interact with each other in a network. Additionally, the tool makes available textual formats of attack paths. The reasoning engine also scales well (O(n^2)) with network size. It works with Nessus and OVAL vulnerability scanners. The tool works command-line user interface.
- Attack Graph Toolkit is an open source developed by Oleg Sheyner at Carnegie Mellon University (2004). The tool is based on Linux operating system platform and has a graphical interface. It scales poorly with the complexity of the network as it increases exponentially with the number of nodes. The project also seems abandoned as it has not been updated since 2007.

Other examples of attack graph tools include FireMon and SkyBox View. Based on previous research (Yi et al., 2013), MulVAL has an advantage over most attack graph tools, especially when considering availability since it is open source. In the previously mentioned attack graph tools, only MulVAL and Attack Graph Toolkit are open sources. MulVAL makes available resources for further security risk analysis, has a more active community and scales better with the complexity of a network. Considering these factors, MulVAL is the most suitable attack graph tool for this thesis. The next section will elaborate more on this tool.

2.3 MulVAL

MulVAL produces a logical attack graph using the logic Datalog programming language to disclose whether the network has possible threats. More

importantly, it shows in what different ways in which a network can be exploited to gain increased privileges. It creates its graphs based on configurations such as network policies and vulnerability data gathered from scanners as Nessus or OVAL along with vulnerability data from National Vulnerability Database. MulVAL uses logical reasoning to identify possible policy violations through information from the mentioned sources.

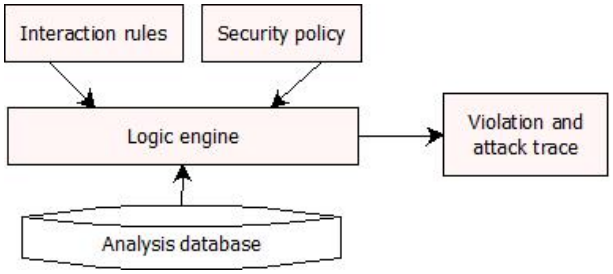

Figure 2.8: MulVAL framework.

Figure 2.8 illustrates the MulVAL attack graph framework in five parts. The logic execution engine collects information about the network such as *interaction rules*, *security policy* and *analysis database*. Given these configurations, MulVAL can simulate the behaviour of an attacker in the network. The logic engine generates graphic and textual format of possible violations and attack paths.

According to the developers of MulVAL (Ou et al., 2005), a vulnerability analysis tool must satisfy two features in order to be useful in practice:

- Be able to integrate formal vulnerability specifications from bug-reporting community.
- Be able to scale with increased network complexity.

MulVAL does satisfy these features as it enriches vulnerability information using the National Vulnerability Database and can generate attack graph in seconds even with a network of thousands of nodes (running time $O(n^2)$ with n being the number of nodes).

Initial network configurations (i.e. available machines, active services, inter-host reachability etc.) in addition to a database of known vulnerabilities make MulVAL able to identify all potential attack paths an attacker can exploit in a network. These paths are united in a logical attack graph proving how successful a potential attack can be based on initial attacks towards for instance the outer edges of the network as the DMZ.

2.3.1 Representation

As illustrated in Figure 2.2, a vulnerability scanner such as OVAL or Nessus can detect a vulnerability such as CVE-2010-0425 (Table 2.1) locatad on

a webserver. However, it cannot associate the effect of the vulnerability such as what are the consequences and how can it be exploited. A vulnerability database (NVD) developed by the National Institute of Standards and Technology (NIST) provides information that enrich the data about identified vulnerabilities. MulVAL converts the information into Datalog clauses such as:

```
vulProperty('CVE-2010-0425', remoteExploit,
                privEscalation).
```

The vulnerability enables an attacker to execute arbitrary code on the webserver. Extracted information about a host collected from a scanner, such as a webserver, is converted into Datalog entries in the following way:

```
networkServiceInfo(webServer, httpd,
                   TCP, 80, apache).
```

The collected properties about the webserver states that it is running a *httpd* daemon, uses TCP and listening on port 80, the user is named *apache* and represents the user privilege the daemon has on the machine. Network configurations are in MulVAL interpreted as abstract host-access control lists (HACL). It specifies all allowed accesses between machines in the network and are normally controlled by packet control mechanisms such as routers, switches and firewalls. The following example shows how Datalog stores network configurations where the webserver is available for connection from the internet using TCP port 80:

```
hacl(internet, webServer, TCP, 80).
```

Other entries include principal bindings such as *hasAccount*, binding principal user information such as administrator access to network hosts:

```
hasAccount(sysAdmin, Host1, root).
```

Furthermore, policy declarations can describe which principal are allowed access what data. If a policy is not declared as allowed, it is regarded as probibited:

```
allow(sysAdmin, write, webPages).
```

There are numerous additional entries MulVAL reads from Datalog. The following list describes some of them:

Entry	Description
clientProgram	Describes the privilege of a client program when it is executed and on what machine
setuidProgram	Defines a setuid executable on the system and its owner
nfsPath	Specifies the owner of a particular path in the file system
vulExist	Defines what vulnerability is available on a specific machine and what program can be exploited.
execCode	Possible outcome of an attack. An attacker might gain privilege to execute code on a given machine with the privilege of a specified user.

Table 2.2: Examples and descriptions of MulVAL Datalog entries (Ou, Govindavajhala, & Appel, 2005).

The information MulVAL processes can be organized into three different entities shown in an attack graph: *configuration* entities, *exploit* entities and *privilege* entities.

- *System configurations* are represented as rectangular shapes. These include host access permissions, existing vulnerabilities, applications, etc.

- *Privileges* are represented as diamond-shapes. A privilege is what an attacker can gain through exploits.

- *Exploits* are represented as elliptical shapes. A potential exploit links the pre-conditional configuration entities, which enables the exploit, with the effect of the attack, the post-conditional privileges (Homer & Ou, 2009).

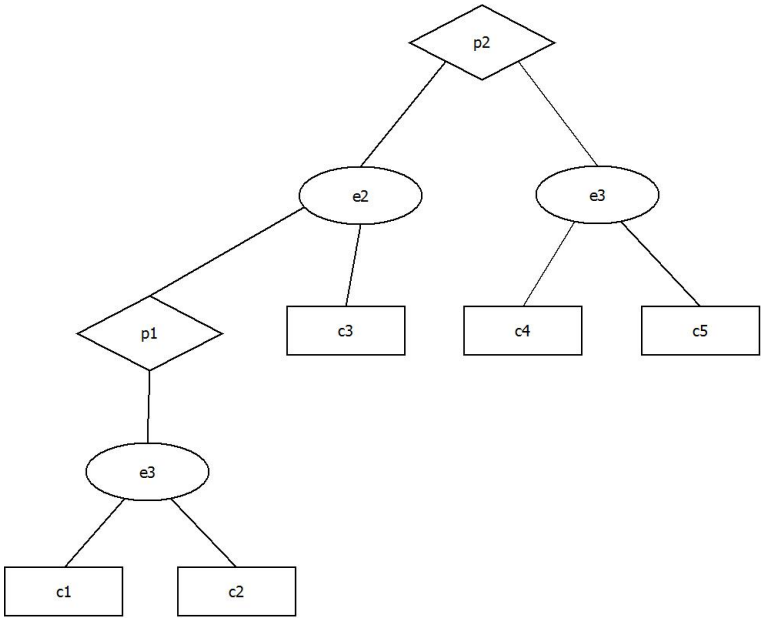

Figure 2.9: MulVAL attack graph.

Arcs coming out from an exploit entity form logical AND relations, meaning all of the child relations must be true in order for the exploit can be used. Removing only one of the child relations would be sufficient for the exploit to be unavailable. In the MulVAL attack graph in figure 2.9, we can express the exploits in the following boolean formulas:

$$\begin{aligned} e1 &= c1 \wedge c2 \\ e2 &= p1 \wedge c3 \\ e3 &= c4 \wedge c5 \end{aligned}$$

Arcs from privilege entities like *p1* and *p2* form logical OR relations. This indicates that only one of its child relations need to be true in order for a privilege to be achieved. Either *e2* or *e3* is required for obtaining privilege *p2*.

$$e1 = e2 \vee e3$$

2.4 Network Hardening

In security communities, network hardening has often been considered as an art rather than science (Wang, Albanese, & Jajodia, 2014). Experienced

security analysts do tedious work to identify and prioritize different vulnerabilities and network weaknesses for fixing and patching. To make network hardening more like science, rather than art, systematic approaches to automatically compute potential hardening solutions are essential.

Network and system administrators often don't care about attack sequences, but rather determine best way to harden their network. A straightforward set of network hardening options which can provide a security guarantee for a given resource is needed. However, a guarantee for security might not always be an option. Fully hardening a network may require such a high cost that it outweighs the security risk. Shutting down the entire network will fully harden the network, but the the cost exceeds the security risk. Previous research have dealt with network hardening and some approaches have inspired this thesis, especially network hardening with respect to initial network conditions.

Attack paths can help network and system administrators with network hardening, and in order to increase security, all attack paths must be accounted for. Tools based on attack graphs can reveal multi-stage attack opportunities in a seemingly well protected network by enumerating all possible attack paths. To optimize this process, a representation that takes all possible attack paths into account, but does not necessarily enumerate all of them is needed. For instance, it is sufficient to know that a particular exploit is required for all possible paths, without necessarily generating all of them. The initial network configurations are the conditions to consider when hardening the network, more explicitly the exploit preconditions in a network.

2.4.1 Related work

For network hardening, we need to measure the overall security of a network. Previous research (Wang, Singhal, & Jajodia, 2007) states that a crucial part in measuring network security lies in understanding the interplay between network components such as how vulnerabilities can be combined by attackers in an advancing multi-stage attack. The framework in this study focuses on computing overall security with respect to critical resources. Two dependency models are presented, one captured by attack graphs, and another captured by additional functions. The latter affects the measure of network components but does not enables it to be reachable. Another approach measures the security strength of a network in terms of using *the weakest attacker* model, that is the weakest adversary who can successfully penetrate the network (Pamula, Jajodia, Ammann, & Swarup, 2006). Other works measure how likely a software is vulnerable to attacks using a metrics which is called *attack surface* (Howard, Pincus, & Wing, 2005) (Manadhata, Wing, Flynn, & McQueen, 2006). That is a security metric for comparing the relative security of similar software systems where the *attack surface* is an indicator of the software's security.

Network hardening with respect to initial conditions level was introduced in 2003 (Noel, Jajodia, O'Berry, & Jacobs, 2003). The article states that an approach using configuration elements is beneficial compared to exploit-level approaches as it resolves hardening irrelevancies and redundancies better. The conducted research was to find a set of initial conditions that can disable a goal condition with a minimum cost. The authors represent the resources in a network as logical propositions of initial conditions where vulnerabilities are viewed as Boolean variables. A false condition would mean the condition is suggested disabled for hardening. The presented product does not scale well as the number of terms in the equation can grow exponentially in the number of conditions in the network.

Another minimization analysis approach has been proposed by Homer and Ou (2009). They introduce a methodology where security requirements can be interpreted as a Boolean formula. The research presented by Homer and Ou propose two SAT solving techniques, namely MinCostSAT and UNSAT Core Elimination. The

- MinCostSAT: Minimizes cost by utilizing user-provided discrete cost values to find mitigation solution.

- UNSAT Core Elimination reduces complexity in reconfiguration to simple choices between conflicting requirements. Previous policy decisions of the human user are placed in a partial-ordering lattice in order to further reduce the choices.

This approach reduces complex problems to manageable proportions in addition to requiring minimal user interaction in order to rapidly fix misconfigurations which can lead to multi-stage attacks. The human interaction includes making decisions about relative value of specific instances of security and usability making the research able to account for both security and usability requirements. Results prove that this approach is scalable and effective.

Research conducted by Noel et al. (2003) introduces a framework for computing the minimum-cost solution with guaranteeing the safety for given critical resources. Also here, minimization is done at the level of initial conditions. The proposed solution extends a graph-based representation of exploit dependencies. The representation in the study has low-order polynomial complexity in contrast exponential complexity which have been found in most found works according to the authors.

2.5 Learning Automata and Population-Based Incremental Learning

2.5.1 Learning Automata

Learning Automata (LA) is a self-adapting machine and was initially introduced by Narendra and Thathachar (1974). The goal of this mechanism is to "determine the optimal action out of a set of allowable actions, where the optimal action is defined as the action that maximizes the probability of being rewarded" (Oommen & Agache, 1999). Learning is defined as a permanent change in behaviour based upon previous experience. A learning system is therefore characterized by its ability to consistently improve its behaviour with the goal to reveal an ultimate solution.

Learning Automata is often used in systems where the knowledge about the environment they work in is limited. An environment could be unpredictable because it is changing over time. This is in academic or professional fields called a *stochastic* environment. *Stochastic Learning Automata* is used to increase the probability that an action will succeed in such an environment. The LA mechanism uses stochastic optimization to learn about the environment. A random generated action is selected based on a probability vector, after which the probability actions are changed based upon how the environment is responding, then the operations are repeated. The automaton uses the response and previous knowledge from past iterations to determine the next action. Over time, LA learns to choose the right action, adapting with the environment(Agache & Oommen, 2002).

LA is provides problem solving which is applicable in numerous fields, including networking and communications where problems such as distributed scheduling (Seredyński, 1998), neural network adaptation (Meybodi & Beigy, 2002) and intelligent vehicle control (Unsal, Kachroo, & Bay, 1997) have been solved using this mechanism.

A LA is defined by a quintuple $\langle A, B, Q, F(.,.), G(.)\rangle$ (Narendra & Thathachar, 2012), all of which are described in Table 2.3:

$A = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$	The set of outputs/actions the LA has to choose from. $\alpha(t)$ is an action chosen for any instant t.
$B = \{\beta_1, \beta_2, \ldots, \beta_m\}$	The set of input to the automaton where $\beta(t)$ is the input of any instant t. β can be either infinite or finite, but is commonly $B = \{0, 1\}$ where $\beta = 1$ represents a penalty, meanwhile $\beta = 0$ represents a reward.
$Q = \{q_1, q_2, \ldots, q_s\}$	the set of finite states, where $Q(t)$ denotes the state of the automaton at any instant t.
$F(.,.) : Q \times B \mapsto Q$	A mapping function, called the transition function, that maps the current state and input to the next state at the instant t, such that, $q(t+1) = F[q(t), \beta(t)]$. It determines the state of the automaton at any subsequent time instant $t+1$.
$G(.)$	is a mapping $G : Q \mapsto A$. It is called the output function, it maps a current state into the current output. G determines the action taken by the automaton if it is in a given state as: $\alpha(t) = G[q(t)]$.

Table 2.3: Quintuple, Learning Automata

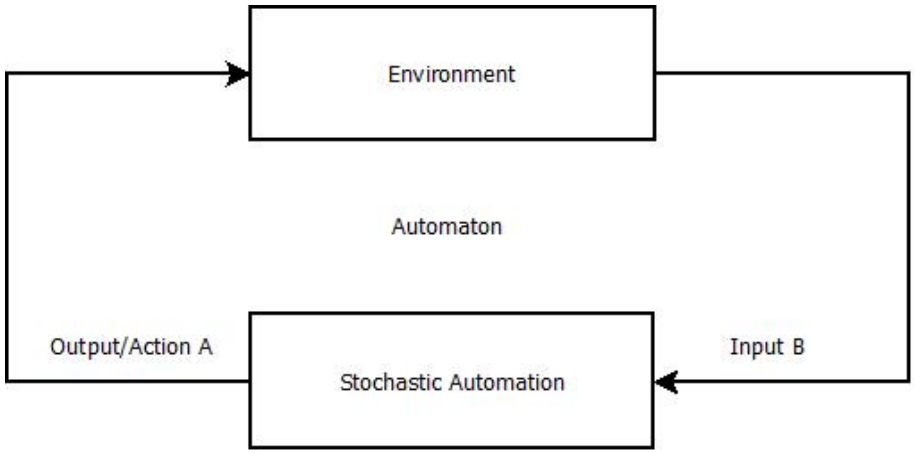

Figure 2.10: Learning automaton.

The learning process is based on repeating iterations where LA continuously interacts with an environment and processes responses to its actions. After trying numerous actions, LA learns the most optimal action. While running, the automaton suggests a set of actions, but is constrained to choose only one of them. When the action is chosen, the environment responses $\beta(t)$ at time t to this chosen action. The action is either rewarded or given a penalty. Based on $\beta(t)$, the state of the automaton is updated, continued by selecting a new action at $(t+1)$.

2.5.2 Population-Based Incremental Learning

Population-based incremental learning (PBIL) is a generalisation of Learning Automata and is a probabilistic model commonly used for optimization of large dynamic combinatorial search problems. It is a combination of evolutionary optimization (Fogel, 1994) and hill climbing (Gent & Walsh, 1993) according to Baluja (1994). The method incorporates genetic algorithms (GA) and competitive learning for function optimization, but rather than being based on population-genetics, PBIL is similar to learning automata in which the automata chooses actions independently. The combination of these two methods constitute a tool much simpler than a GA and outperforms GA on many optimization problems. The intention of the algorithm is to generate a real valued probability vector which, when sampled, reveals high evaluation solution vectors with high probability (Baluja & Caruana, 1995).

The features of PBIL are as follows (Folly, 2005):

1. PBIL has no crossover and fitness proportional operators.
2. It works with probability vectors, normally from 0-1, which control the random bitstring generated by the algorithm and used to produce others through learning.
3. All solutions in the population are not stored, only the currently best solution and the one being evaluated.

Evolutionary Algorithms (EA), which utilize the principles of natural selection and population genetics (e.g. GAs), have become common for optimization and search techniques due to their powerful capabilities for finding solutions to difficult problems. Especially in static environments, where the landscape does not change during computation (Golberg, 1989). However, real-world environments are often prone to changes, making traditional EAs unsuitable as they cannot adapt properly to changed environments once converged (Yang & Yao, 2005). In contrast, PBIL has shown itself to be very successful when compared to different standard genetic and hill climbing algorithms on various benchmarking (Baluja & Davies, 1997) and real-world (Greene, 1996) problems. Moreover, theoretical work on this method's convergence behaviour has been carried out (Lozano, 2000)(Hiihfeld & Rudolph, 1997).

For each iteration, a set of samples are generated according to the current probability vector. The set of samples are evaluated respectively to the problem-specific function. In the end, the probability vector is learnt and shifted towards a solution with the best result. The distance the probability vector is pushed each iteration depends on a parameter called the *learning rate* (LR) which weights the previous vector with the new vector. The learning rate is commonly set to be 0.05. For the next iteration, when the probability vector is updated, a new set of solutions are generated according the the updated probability vector. The algorithm ends when

the termination conditions are satisfied, usually when all vectors have converged (normally towards 0 or 1) or the number of iterations have reached a maximum number.

As an example, the solution to a problem can be represented as a string of 1's and 0's. The initial values of a probability vector is normally 0.5 and a satisfactory final probability vector should be 0.99, 0.01, 0.01, 0.01, 0.99, 0.99. Sampling from such an initial vector reveals random solution vectors since there is an equal probability of generating 0 or 1. As the search progresses, the values in the probability vectors will shift towards 1 or 0. The pseudo code for PBIL is examplified in Listing 2.1.

Listing 2.1: The basic version of the PBIL algorithm for a binary alphabet, adapted from Baluja and Caruana, 1995.

```
****** Initialize Probability Vector ******
for i :=1 to LENGTH do P[i] = 0.5;

while (NOT termination condition)
    ***** Generate Samples *****
    for i :=1 to NUMBER_SAMPLES do
        solution_vectors[i] := generate_sample_vector_according_to_probabilities (P);
        evaluations[i] :=Evaluate_Solution (solution_vectors[i]);

    solution_vectors = sort_vectors_from_best_to_worst_according_to_evaluations ();

    **** Update Probability Vector towards best solutions****
    for j :=1 to NUMBER_OF_VECTORS_TO_UPDATE_FROM
        for i :=1 to LENGTH do P[i] := P[i] * (1.0 - LR) + solution_vectors[j][i]* (LR);

PBIL CONSTANTS:
NUMBER_SAMPLES: the number of vectors generated before update of the probability vector (200).
LR: the learning rate, how fast to exploit the search performed (0.005).
NUMER_OF_VECTORS_TO_UPDATE_FROM: the number of vectors in the current population which are
used to update the probability vector (2).
LENGTH: number of bits in the solution (determined by the problem encoding).
```

The initial probability vectors are established. The new sample vectors are generated followed by an evaluation of them, then sorted from best to worst. The highest evaluation vectors are kept. The learning rate (LR) determines how fast the probability vectors shifts each iteration. The probability is defined by:

$probabilityVector_i = (probabilityVector_i \times (1.0 - LR)) + (LR \times vector_i)$
where $probabilityVector_i$ is the probability of generating a 1 in bit position $_i$ and $vector_i$ is the i position in the solution vector which the probability vector is shifted towards. LR is the learning rate.

Part II

The project

Chapter 3

Approach

As an attempt to answer the problem statement, this thesis will make use of all aspects described in Chapter 2 to solve a constrained optimization problem with respect to network hardening. This thesis' ambition is to develop a tool which the goal is to analyze the current network security state, based upon data from an attack graph, and be able to compute a solution which will provide increased network security. More specific, the tool should suggest what configurations should be fixed or patched. However, one important factor is *what* or *where* we regard as a threat. A potential attack could be initiated from anywhere, from the Internet or even from any computer inside the network itself. Nonetheless, this thesis assume the network and system administrator wants to secure the network with respect towards attacks from outside the boundaries of the network, in most cases the Internet. Moreover, the cost of fixing/patching configurations varies. This research does not take into account the cost variations of configurations but assume they all have same cost. Additionally, this research does not take into account the severity of different vulnerabilities, neither the likeliness of a vulnerability being exploited.

The main goal in this study is to reduce the number of attack paths as many as possible using PBIL, and the solution shall take usability into account along with security. The reduction of attack paths will be illustrated as a *reduced attack graph*. The reduced attack graph can be compared with the initial attack graph in order to observe to what degree the graph has been reduced. The initial attack graphs are added as appendices due to their complexity. The goal is to be able to compare the reduced size of the graphs, and due to the complexity of some graphs, details might not be readable on printed paper. However, using PDF-readers provide scalability beneficial for examining the details of the graphs.

The thesis will present two approaches both based on PBIL, one *acceptance-rejection* approach, and the other is a *penalty* approach. These approaches will be described later in this chapter. Figure 3.1 illustrates the desired solution. The parallelograms represent the tool developed in this study.

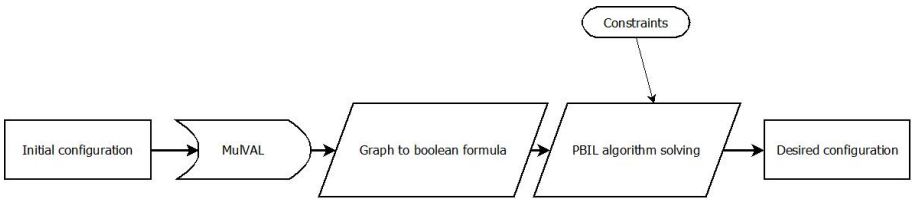

Figure 3.1: Data flow.

The attack graphing tool used in this research is MulVAL attack graph. MulVAL is open source and uses external sources to enrich vulnerability data. Additionally, MulVAL uses a command-line interface which is useful for fully automated graphing solutions. The tool provides csv-files which are easy to understand and use while programming. This approach to network hardening could use any attack graphing tool which provide data files for importing into scripts, and in this case, MulVAL has shown itself to be very satisfactory compared to others for this study. For solving a combinatorial optimization problem with respect to initial network conditions, this research has found the PBIL algorithm suitable.

We would like to maximize the network security while having a constraint: the number of configurations in a network that can be activated or cannot be deactivated. Standard PBIL method itself does not have the ability to solve constrained optimization problem. In order to mitigate this limitation, we rely on two algorithms, namely *Acception/Rejection* and *Penalty* based PBIL. Chapter 3.2 will give an introduction to these two methods for constrained optimization.

3.1 Measuring security

When evaluating samples in the computations, a formula for measuring security is needed. A common first logical thought would be to evaluate security in terms of the number of vulnerabilities an attacker from the Internet are able to exploit. However, when evaluating security, what is important to consider is the consequences we wish to avoid. In terms of an attack graph, an exploit leads to a privilege, and a privilege can be considered as a consequence (POST-condition) of realizing an exploit. As described in Chapter 2.2 and illustrated in Figure 2.5, multiple exploits may lead to the same privilege, they namely form an OR-relation. Fixing/patching a vulnerability may not be sufficient to avoid the consequence/privilege, as another vulnerability might lead to the the same. In such a case, an attacker would just use another exploit in order to reach a desired privilege. Therefore, evaluating security in terms of total number of privileges an attacker from the Internet is able to gain is suggested.

3.2 Constrained optimization

To solve a problem with constraints, we wish to apply two approaches: the *Acceptance–Rejection* and the *Penalty* approach. These methods are basic techniques for generating observations from a distribution. They are for instance common in experiments using the Cross-Entropy method (CE) which was presented in 2004 by Rubinstein and Kroese. The CE method is widely applied in discrete optimization tasks and has similarities to PBIL.

Acceptance-Rejection (AR) is a method for simple constraints. Suppose we generate a random vector from a normal distribution, we can then accept it if the vector falls in the interval of interest, or otherwise reject it. One important note when using the AR method, the threshold for acceptance should not be too high, or else it would cause rejection of too many samples which in turn leads to time consuming computations. However, when using this method correctly, it reduces the number of evaluations to satisfactory samples only.

The *Penalty* is more generally applicable and quite easy to implement. The approach involves using a penalty parameter which is dependant on the original problem. The parameter can be, to some degree, a bit arbitrary, and what is important to be aware of is that the solution to the problem can be very sensitive to this parameter.

3.3 Constraints

Some human interaction is needed for hardening the network with constraints. In the suggested solution, we can divide the constraints into two parts.

3.3.1 Maximum configuration constraint

The user of the tool should decide a threshold for how many configurations the tool should suggest fixed or patched. In a network of 100 exploit-related configurations, a network and system administrator would for instance like to know which 10 configurations should be fixed in order to provide the best security. In an optimal network, all vulnerable services should be patched at all times, but in real-life, this is not the case. Having this in mind, a network and system administrator must prioritize when hardening the network. He wants to know which immediate fixes he should do in order to increase security in his network.

As an example, in a network of 145 exploit-related configurations $c = (c_1, c_2, c_3, c_4 ..., c_{145})$, the network and system administrator knows that he has limited time to secure his network. He knows that within the

coming day, he must increase the security dramatically. He also knows that he can fix/patch approximately ten configurations within this time limit. He should therefore be able to tell his network hardening tool to suggest a maximum of ten configurations that should be fixed or patched. Thus, the tool will present ten configurations that, while TRUE, leave the network open to the biggest threat. The same configurations provide the best possible security while FALSE. Such an suggestion output could be: $(c_5, c_{14}, c_{12}, c_{39}, c_{52}, c_{67}, c_{91}, c_{102}, c_{138}, c_{144})$. The two approaches *acceptance-rejection* and *penalty* are used to handle this type of constraint. How this is implemented is described more thoroughly in Chapter 3.4.3.

3.3.2 Configuration disqualify constraint

The user of the tool should also be able to disqualify configurations which should not be suggested fixed. Configuration examples include the fact that an *attacker is located on the Internet*, that a *service is available from the Internet*, a *service is running* or a *service has a vulnerability*. A network and system administrator cannot help the fact that potential attackers are on the Internet, which makes it impossible to fix. Additionally, enterprises often rely on offering their services to the public through Internet. In most cases, shutting access from the Internet towards its services is not an option. Configurations representing available vulnerabilities should always be able to be suggested fixed, but in some cases there is no patch or fix available for that vulnerability in particular. The user should therefore be able to tell the tool that such configurations cannot be suggested fixed by the tool.

Given a network of ten exploit-related configurations $c = (c_1, c_2, c_3, c_4..., c_{10})$, the administrator should be able to give specific configuration constraints. An example of such an constraint could be $\beta = (c_1, c_2, c_8)$. The solution should never suggest to fix/patch any of the stated configurations β.

3.4 Planning process

This section describes how the planning process towards the tool. First, necessary notations are established, followed by selection of programming language. Lastly, the suggested approach when programming the tool is displayed.

3.4.1 Notations

Given a list of exploit-related configurations:
$c = (c_1, c_2, c_3..., c_N)$.
PBIL can generate samples with length N:
$x = (x_1, x_2, x_3...x_N)$ where $x_i \in \{0, 1\}$

$$x_i = \begin{cases} 1 \; if \; c_i \; is \; enabled \\ 0 \; if \; c_i \; is \; disabled \end{cases}$$

Enabled configurations are exploit-related configurations which are *turned on*, also regarded as *not suggested patched/fixed*, *TRUE configurations* or in the code read as 1. Disabled configurations are exploit-related configurations which are *turned off*, also regarded as *suggested patched/fixed*, *FALSE configurations* or in the code read as 0.

$f = \sum I(x_i = 0)$ represents the number of disabled exploits where $I(.)$ is the indication function.
PBIL generates samples based on probability vectors:
$p = (p_1, p_2, p_3..., p_N)$ where $p_i = probabilityVector_i$ described in Chapter 2.5.2. p_i is the probability that $x_i = 1$.

3.4.2 Programming Language

The suggested tool will be written in the the programming language *Python*. Python is efficient for file management and has numerous libraries for efficient programming, making it ideal for this project. Additionally, python is widely known for its simple syntax which makes the code easy to read and comprehend.

3.4.3 Suggested approach

The suggested approach for programming is to follow the concept of PBIL. The input data will be retrieved from MulVAL's generated csv-files, namely *VERTICES.CSV* and *ARCS.CSV*. The initial configurations, exploits and privileges can be retrieved from *VERTICES.CSV*. The following excerpt is an example of this file:

Listing 3.1: VERTICES.CSV.

```
3,"canAccessHost(citrixServer_2)","OR",0
4,"RULE 8 (Access a host through executing code on the
    machine)","AND",0
5,"hasAccount(victim_2_2,citrixServer_2,user)","LEAF",1
6,"principalCompromised(victim_2_2)","OR",0
7,"RULE 12 (password sniffing)","AND",0
8,"RULE 3 (remote exploit for a client program)","AND",0
9,"accessMaliciousInput(citrixServer_2,victim_2_2,ie)","OR
    ",0
10,"RULE 22 (Browsing a malicious website)","AND",0
```

Each line has four comma-separated values. The third value reveals what type of entity the line serve as: **LEAF** are configurations, **AND** are exploits and **OR** are privileges. The first value is an unique identifier (ID) of the entity. The *ARCS.CSV* file represents the relations between the different entities:

Listing 3.2: ARCS.CSV.

```
4,1,-1
3,4,-1
2,3,-1
2,5,-1
7,1,-1
```

The two first values on each line correspond to a relation between two entities. On line two we see that there is a relation between the entity with the unique ID 3 and 4. Looking back at Listing 3.1, we know that there is a relation between *3,"canAccessHost(citrixServer_2)* and *RULE 8 (Access a host through executing code on the machine)*. It means that by exploiting vulnerability ID 4, we gain the privilege with ID 3.

When importing the data, the suggested solution is to translate them into the following lists and directories in python:

- A list containing the ID of all the configurations. Such a list will be useful for knowing which ID number is a configuration, and also useful for when setting the initial configurations.
- A list containing the ID of all the exploits.
- A list containing the ID of all the privileges.
- A dictionary of all the exploits and their corresponding AND-relations.
- A dictionary of all the privileges and their corresponding OR-relations.
- A dictionary of all the exploits and the privilege they lead to.

When running the script, it should count the number of configurations and create a probability vector list where the number of vectors correspond the the number of configurations and set all initial values to 0.5. If a graph show 6 initial configurations, the initial probability vectors should be a list *[0.5,0.5,0.5,0.5,0.5,0.5]*.

in the continuation, the suggested program should work in a following loop, where one lap in the loop is one iteration:

Listing 3.3: Iteration loop.

```
converged=False
count=0
while (converged == False) and (count < max_iterations):
        genNum=generateVectors()
        avg=findBestLists(genNum)
        sample=updateProbVector(avg)
        converged=hasConverged(sample)
        count=count+1
```

- Line 3: Continue to next iteration as long as the probability vectors have not converged or reached the highest number of iterations given.

- Line 4: Generate a given number of samples with random numbers in a 0-1 interval for each probability vector. If the generated number is less than the corresponding probability vector, then regard the configuration to be TRUE (1), otherwise regard the configuration as FALSE (0). Check whether a disqualified configuration is FALSE, and change it to TRUE. While using the *acceptance-rejection* method, disregard samples that exceeds a given number of FALSE configurations (threshold t). Stop generating samples when there are X number of accepted samples. X is recommended to be 100.

- Line 5: Evaluate each sample according to the measured security as described in section 3.1. Let measured security be m, where optimal security m is 0. When using the *penalty* approach, add the penalty p to the number of privileges an attacker can realise (m). The suggested approach is to *only* add a penalty to samples that exceeds a *threshold* (t) of FALSE configurations (f) in a sample. Let $p = \lambda \times (f - t)$. The penalty parameter λ and threshold parameter t should be established based on the total number of configurations given in the attack graph to ensure scalability. In both approaches, *acceptance-rejection* and *penalty*, security s is measured by $s = m$, meanwhile samples penalised in the *penalty* approach by exceeding $f > t$ are calculated $s = m + \lambda \times (f - t)$. The ten best results from the total number of 100 samples are recalculated into one sample where the mean of every $value_x$ from the ten best samples are kept as shown in Figure 3.2.

- Line 6: Now the newly generated sample (called the *elite* sample) can be weighed with the previous sample. The learning rate (LR) indicates how fast the algorithm learns and the $freshSample$ can be calculated as $freshSample_i = oldSample_i \times (1 - LR) + newSample_i \times LR$. The suggested LR should be 0.05 or 0.1.

- Line 7: The sample is checked whether it has converged or not. It has not converged if a $sample_i > 0.01$ AND $sample_i < 0.99$.

Notation	Description
m	Number of privileges an attacker can gain from a given configuration
f	Number of FALSE configurations in a sample. $f = \sum I(x_i = 0)$ described in 3.4.1.
t	Threshold. The threshold states a point of undesired amount of FALSE configurations.
λ	The penalty parameter.
p	Penalty. Penalty $p = \lambda \times (f - t)$.
s	Calculated security. *Acceptance-rejection* approach: $s = m$. *Penalty* approach: $s = m + p$.

Table 3.1: Notations

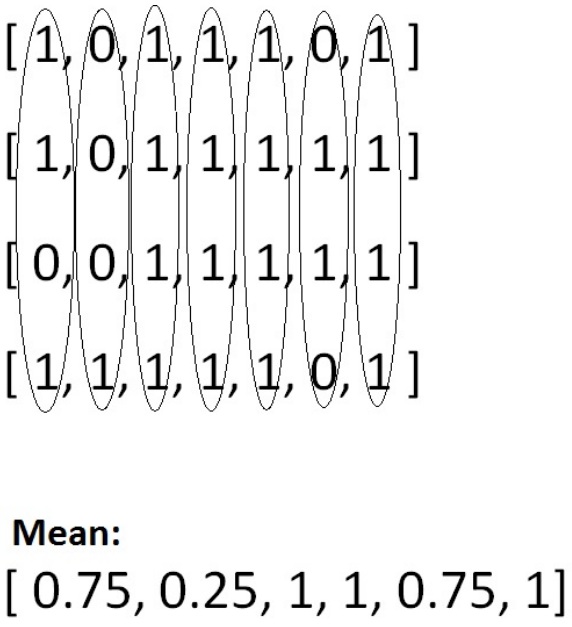

Figure 3.2: Mean of $value_x$ from best samples.

3.5 Example case

Figure 3.5 is an example of a very simple attack graph. The graph is based on a simple network of three nodes. In the network, an attacker has been identified on the Internet (ID 18), and he has access to the webserver (ID 17). If the network and system administrator wanted complete security from the Internet, he would deny all access from it by removing configuration ID 17.In that case, attackers would not gain any privileges on the network and his work would be done. This is in most cases not an option, and the user of the tool will have to disqualify ID 17 and 18. This tells the tool

to never suggest fixing or patching these configurations. The following figures show how the graph will be reduced to when when disqualifying different options.

Disqualifying ID 18 and 17. The tool should suggest to fix/patch configuration ID 19 or 20 as both configurations lead to same exploit. Fixing one of them is sufficient for securing the network towards further multi-stage attacks.

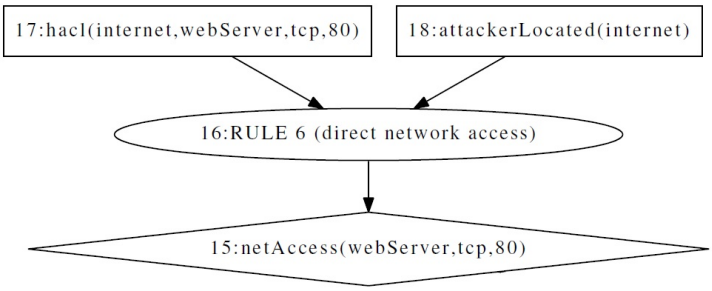

Figure 3.3: Attack graph reduced when disqualifying ID 17 and 18.

Disqualifying ID 17,18,19,20. ID 19 represents a service (apache) running. The system administrator must have this exact service running and disqualifies it. ID 20 represents a vulnerability. The administrator cannot find a patch for this vulnerability so he disqualifies it. The tool should suggest to fix/patch configuration ID 12 plus 24 or 25.

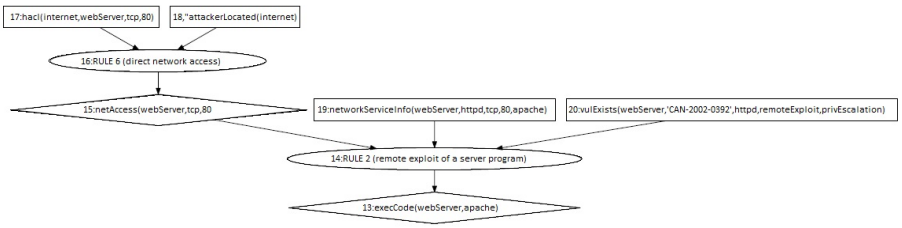

Figure 3.4: Attack graph reduced when disqualifying ID 17,18,19,20.

From the example experiments, we see what the initial attack graph is reduced to when giving various configuration constraints. It is easy to reduce the attack paths in a network of just a few paths as given in Figure 3.5, however, the work becomes more tedious when the complexity of the network increases.

3.6 Experiments

The experiments conducted in this study will include three different networks, all of which will have different constraints in terms of initial

Figure 3.5: Simple attack graph, retrieved from MulVAL's installation package

configurations. The networks represent a variety of complexities. Their initial attack graphs are generated in MulVAL using manually written input files. It means that the graphs are not generated based on a real-world network.

Complexity	Description
Low	Nodes: 6 Exploits: 21 Configurations: 20 Privileges: 18 The network itself is not complex. Two nodes are connected directly to the internet through the perimeter firewall. The four other nodses are connected to an internal router, which in turn is connected to the perimeter firewall.
Medium	Nodes: 8 Exploits: 17 Configurations: 16 Privileges: 14 The network is more complex than the previous one and has two firewalls, one perimeter firewall and one internal firewall. However, the number of configurations and exploits are lower than the previous one.
High	Locations: 2 Subnets: 14 Exploits: 33 Configurations: 31 Privileges: 28 The network is mirrored on two different locations, both connected to the internet. Each location has 7 subnets, constituting a total of 14 subnets. Each location has 3 firewalls and one router.

Table 3.2: Overview of networks used in experiments

Illustrations of suggested network topologies

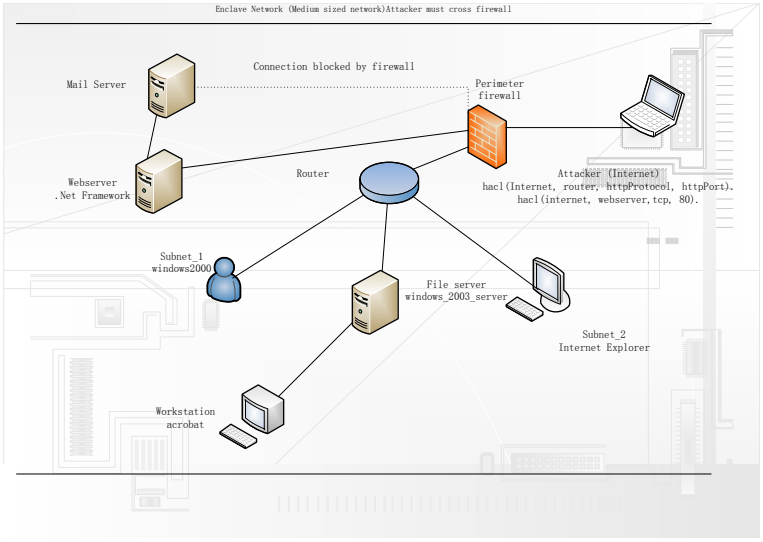

Figure 3.6: Topology network 1.

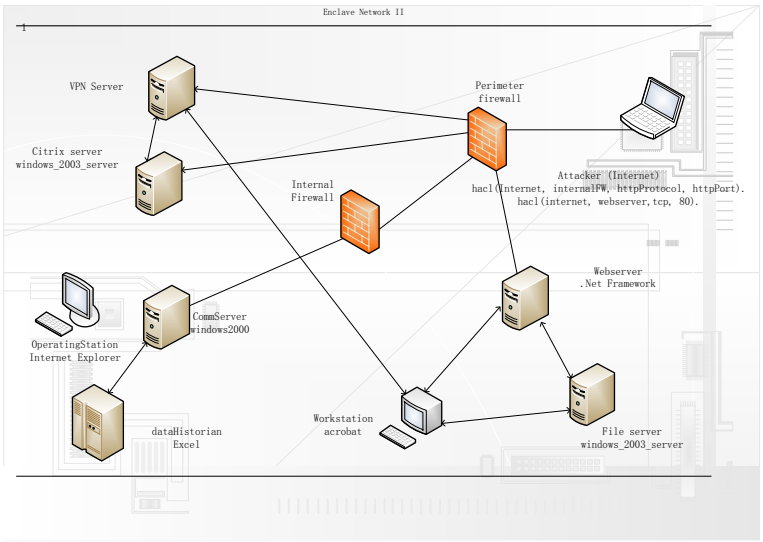

Figure 3.7: Topology network 2.

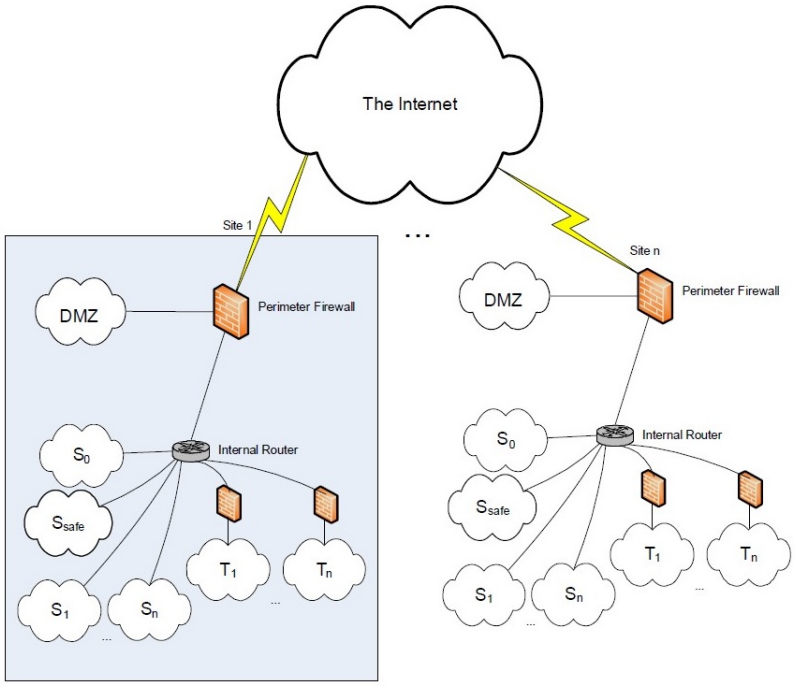

Figure 3.8: Topology network 3.

3.7 Result data

The results should provide sufficient data for answering the following questions:

1. Is PBIL suitable for this exact type of computation?
2. How does the learning rate impact a computation?
3. How does the *threshold* parameter impact evaluations?
4. Can we compare the two approaches, *acceptance-rejection* and *penalty*?

Part III

Results and Conclusion

Chapter 4

Results

This chapter will describe how the tool was programmed. Further on, experiments on how PBIL can reduce the number of attack paths is carried out in addition to experiments directed against the approaches *acceptance-rejection* and *penalty*.

4.1 Implementation

This section will describe how the different methods described in Listing 3.3 work in depth. Examples from source code are shown where deemed appropriate.

4.1.1 Generating samples

The method *generate* has three parameters, the current probability vector sample, maximum number of samples to be generated and a list of configurations which cannot be FALSE/0 (user constraint). The method generates a sample of random numbers between a 0-1 interval where each number is compared to the corresponding value in the current probability vector sample. If the generated value is less than the current value, it gets defined as 1 and 0 if more. Lastly, the sample gets evaluated in a method *satisfactoryVectors*, which will be described in the next paragraph.

Listing 4.1: Generating samples.

```
1 def generate(sample,max_samples,listOfConf,
    cannotDisableConf):
2         iter_Samples=[]
3         while len(iter_Samples) < max_samples:
4                 newSample=[]
5                 for i in sample: #Generate random number
                      between 0 and 1
6                       val=random.random()
7                       if val<=i:
```

```
                                    q=1
                            else:
                                    q=0
                            newSample.append(q)
                    if satisfactoryVectors(newSample,
                        cannotDisableConf): #Disregard sample if
                         it does not satisfy demands
                            iter.append(newSample)
                    else:
                            continue
            return iter
```

The *satisfactoryVectors* method has two functions:

1. If a configuration has landed on value 0 and the user has specified it not to be (user has disqualified the given configuration), the method changes the value to 1. While using the *penalty* approach, the method ends here, it always returns True.

2. While using the *acceptance-rejection* approach, the method counts the number of zeros in a sample and compares it to the threshold t. If the number of zeros in a sample exceeds the threshold, the method returns False, and the sample is discarded.

changes the values to

Listing 4.2: Check sample towards constraints.

```
def satisfactoryVectors(newSample,cannotDisableConf):
        #Make sure all the configurations which user has
            specified not to turn off are turned on
        for idx,val in enumerate(cannotDisableConf):
                if val=='X':
                        newSample[idx]=1

        count=0
        for i in newSample:
                if i==0:
                        count=count+1

        #Not more than a given percent of the
            configurations should be suggested disabled,
            disregard sample otherwise
        threshold=0.2
        if count <= int(round(len(newSample)*threshold)):#
            len(newSample)*0.2: Here you change the percent
            of suggested disabled configurations
                return True
        else:
                return False
```

4.1.2 Evaluate samples

The method *findBestLists* has three objectives:

1. To evaluate every sample, all possible paths to reach a privilege based on the sample data is tested in a loop. If the loop does not find a way to the privilege, it is regarded as unreachable. The *penalty* approach also adds a penalty $\lambda \times (f - t)$ (See Table 3.1 for notation details) if the number of FALSE configurations exceeds a threshold t where the penalty parameter λ and threshold parameter t is specified by the user. Listing 4.3 shows the *penalty* that returns a penalty which is added to the formula. If there is no penalty, it return 0 penalty.
2. Retrieve the 10 best samples based on the evaluations.
3. Calculate and return the elite sample, it is the average of $vector_x$ from the 10 best samples as Figure 3.2 exemplifies.

Listing 4.3: Penalty.

```
def getPenalty(sample):
        threshold=int(round(len(sample)*0.2))
        penaltyConstant=1
        numOfDisabled=sum(x == 0 for x in sample)
        if numOfDisabled > threshold:
                return (numOfDisabled-threshold)*
                    penaltyConstant
        else:
                return 0
```

4.1.3 Update probability vectors

When updating the existing probability vectors, the metod *updateProbVector* takes two parameters, of the newly generated elite sample and the current probability vectors. The learning rate (LR) decides how fast the computation learns each iteration. In Listing 4.4, the RT is 0.1.

Listing 4.4: Update probability vector.

```
def updateProbVector(iterSample,sample):
        #Recalculates the new value of the sample based
            upon the old sample and the new improved
            solution
        learningRate=0.1
        weightNew=learningRate
        weightOld=1-learningRate
        newVal=[]
        for (new,old) in zip(iterSample,sample):
                newVal.append((weightNew*new)+(weightOld*
                    old))
        return newVal
```

4.1.4 Convergence

This method returns FALSE if any of the vectors in the newly updated probability vector has converged. The tool allow one percent tolerance, meaning there has to be at least 0.99 probability of generating a 0 or 1 on all vectors in the next iteration in order to declare all vectors converged. When converged, the iteration-loop stops and the tool can present the findings.

Listing 4.5: Converged.

```
def hasConverged(sample):
        converged=True
        for i in sample:
                if i > 0.01 and i < 0.99:
                        converged=False

        return converged
```

4.2 Experiments

Experiments will be conducted on three suggested networks. Their current attack graphs are added to the appendix. However, due to large graphs, they might not be readable on paper, only in a PDF-reader. The experiments will state the constraints given in the respective experiment. The configuration disqualify constraints are stated as a list of unique configuration ID's. Results will show how the attack graphs are reduced, and how the two approaches act. Result tables are limited to just a few lines in this report, though every experiments are conducted at least 30 times. Mean values are based on the entire experiment population.

4.3 Network 2 experiment

Network 2 is presented first. This network has the least complex attack graph (see Appendix B.1) of the three presented networks.The experiment should show how the tool reduces an attack graph with given constraints.

In this experiment, the network and system administrator needs to harden his network quickly. Therefore, he wants the tool to suggest not more than three configurations which need to be fixed/patched. He determines the constraint *maximum configuration constraint* to three. Due to usability requirements and restrictions, he has the following *configuration disqualify constraints*:

Configuration disqualify constraints
11:attackerLocated(internet):1
25:hacl(commServer,internet,httpProtocol,httpPort):1
12:hacl(citrixServer,internet,httpProtocol,httpPort):1
45:hacl(vpnServer,internet,httpProtocol,httpPort):1
27:vulExists(commServer,'CVE-2010-0483',windows_2000, remoteClient,privEscalation):1
14:vulExists(citrixServer,'CVE-2010-0490',ie,remoteClient, privEscalation):1
46:inCompetent(victim_5):1
13:inCompetent(victim_2):1
26:inCompetent(victim_1):1

Table 4.1: Disqualify constraints network 1

- Configuration 11: Attackers are available on the Internet. The administrator knows he cannot change this.
- Configuration 25,12,45: Due to requirements where keeping services available to the Internet, the administrator decides to disqualify these configurations.
- Configuration 27,14: The administrator knows that patches for these vulnerabilities are not available.
- Configuration 46,13,26: The users in his network cannot be relied on when it comes to security. He takes into account that there is a possibility that the users might perform actions critical to security.

The tool, independent of which approach, repeatedly give the same results: *'5'*, *'19'*, *'47'*.

- 5:hasAccount(victim_2,citrixServer,user):1
- 19:hasAccount(victim_1,commServer,user):1
- 47:vulExists(vpnServer,'CVE-2010-0492',openvpn,remoteClient, privEscalation):1

Fixing/pathching these configurations result in the following attack graph which has reduced the number of privileges an attacker can gain from 14 to 3.

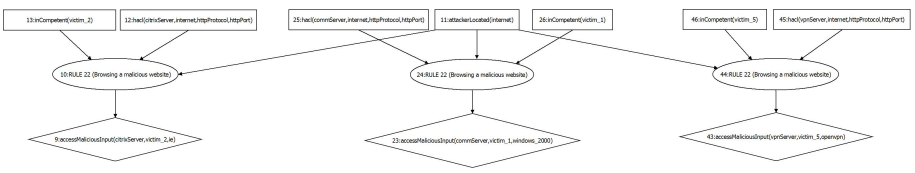

Figure 4.1: Reduced attack graph network 2.

4.3.1 Network 1 experiment

The main goal of this experiment is to observe how the two approaches *penalty* and *acceptance-rejection* compare while having a LR of 0.1 and a LR of 0.05. Another intention is to observe how *accept-rejection* sample generation progresses over iterations. The initial attack graph of the network is in Appendix A.1.

This network has been given the following constraints:

- Maximum configuration constraint: 4
- Configuration disqualify constraint: '11','25','12','38','39','57'

Acceptance-rejection

The maximum configuration constraint implies that all generated samples with more than 4 disabled values should be rejected.

Experiment LR=0.1

Suggested solution	Iterations	Seconds
('13', '26', '40', '58')	65	11.6
('14', '26', '32', '58')	64	11.8
('14', '26', '40', '58')	71	12.8
('14', '26', '40', '58')	75	13.4
('13', '26', '40', '58')	85	15.2
('13', '26', '40', '58')	102	18.1
('14', '26', '40', '58')	90	16.3
('13', '26', '32', '58')	70	12.5

Table 4.2: Results Network 1 Acceptance-rejection LR=0.1

Mean iterations = 77 Mean seconds = 13.9

Experiment LR=0.05

Suggested solution	Iterations	Seconds
('13', '26', '40', '58')	173	30.6
('14', '26', '40', '58')	198	35.9
('13', '26', '32', '58')	175	30.7
('14', '26', '32', '58')	198	34.9
('14', '26', '32', '58')	186	32.9
('14', '26', '32', '58')	164	29.3
('14', '26', '40', '58')	148	26.1
('14', '26', '32', '58')	177	31.7

Table 4.3: Results Network 1 Acceptance-rejection LR=0.05

Mean iterations = 171 Mean seconds = 30.6

Penalty

$\lambda = 1$ and $t = 4$
The threshold value t is how the penalty approach annotates the maximum configuration constraint.

Experiment LR=0.1

Suggested solution	Iterations	Seconds
('14', '26', '32', '58')	69	14.9
('13', '26', '40', '58')	73	16.8
('13', '26', '40', '58')	86	16.5
('14', '26', '32', '58')	75	13.7
('13', '26', '32', '58')	65	12.1
('14', '26', '32', '58')	73	13.5
('14', '26', '40', '58')	77	14.6
('13', '26', '40', '58')	83	15.3

Table 4.4: Results Network 1 Penalty LR=0.1

Mean iterations = 80 Mean seconds = 17.9

Experiment LR=0.05

Suggested solution	Iterations	Seconds
('13', '26', '32', '58')	129	24.3
('14', '26', '40', '58')	179	32.4
('14', '26', '32', '58')	146	26.7
('14', '26', '32', '58')	213	39.4
('14', '26', '40', '58')	152	27.7
('13', '26', '32', '58')	177	32.2
('14', '26', '32', '58')	152	27.9
('13', '26', '32', '58')	156	28.3)

Table 4.5: Results Network 1 Penalty LR=0.05

Mean iterations = 161 Mean seconds = 29.6

We can observe the fact that the various *suggested solution* results within each experiment varies to some degree. However, the experiments between themselves look more or less similar. The reason for this is explained in Chapter 5. However, regardless to which computation output, the reduced attack graph end up identical to Figure 4.2. The hardening reduces the number of accessible privileges from 18 to 2.

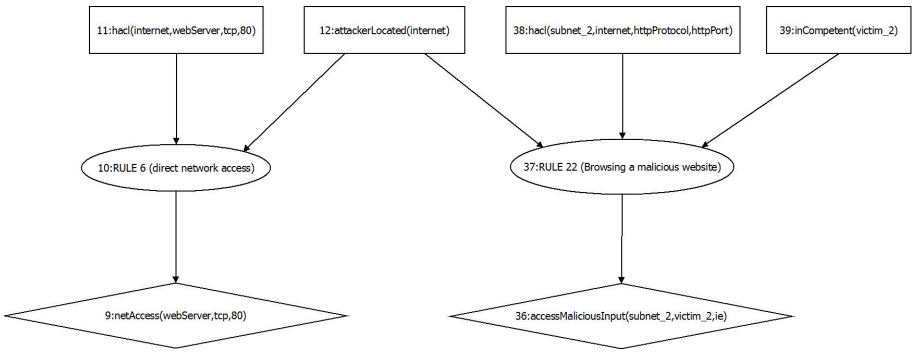

Figure 4.2: Reduced attack graph network 1.

Figure 4.3 illustrates how the *acceptance-reject* approach generates samples per iteration in this experiment.

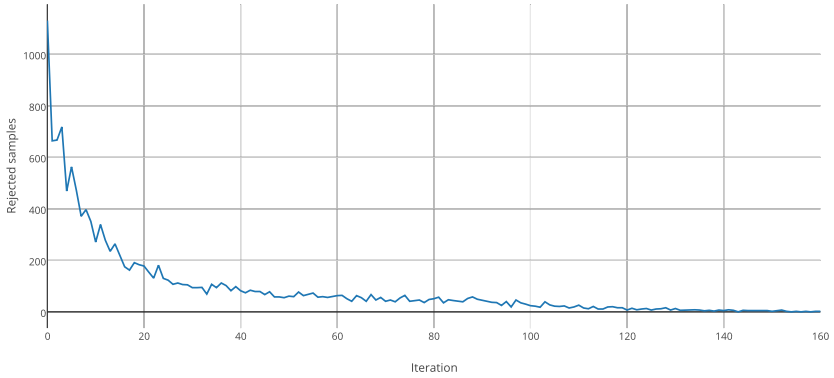

Figure 4.3: Rejected samples per iteration.

Compared to the *penalty* approach, *acceptance-reject* generates around double the amount of samples in a computation with the same amount of iterations. The findings are presented in Figure 4.4.

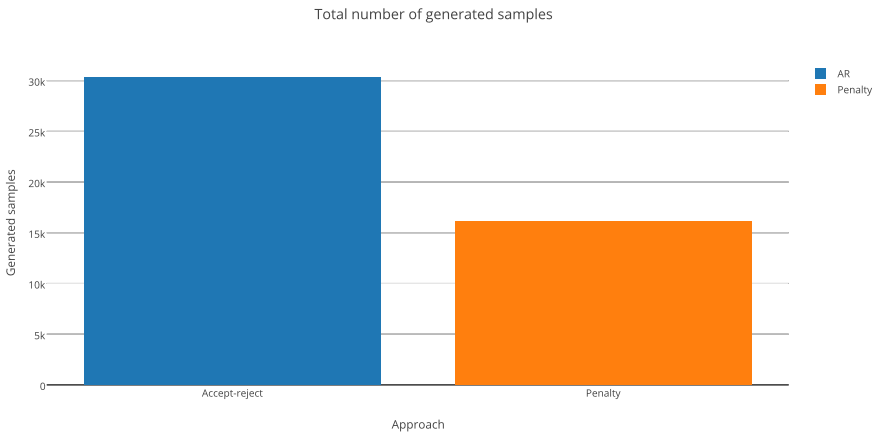

Figure 4.4: Generated samples in one computation.

4.4 Experiment network 3

Experiments in Chapter 4.3.1 show that results between the two approaches vary based on the learning rate. The main goal of this experiment is to observe how the two approaches *penalty* and *acceptance-rejection* compare with respect to time per iteration. Lastly, an experiment on whether increasing the LR dramatically will show deviation in the results is conducted. The initial attack graph is in Appendix C.1.

This network has been given the following constraints:

- Maximum configuration constraint (threshold): 20 % of the configurations.
- Configuration disqualify constraint: '12','13','14','25','26','45', '11','58','59','71','72','90','91','46','47'.

Also in this network, the results in Table 4.6 display that the *acceptance-rejection* approach has an advantage with respect to time with a higher learning rate (0.10), meanwhile *penalty* approach solves the optimization faster with a lower learning rate (0.05).

Mean computation time in seconds

	Acceptance rejection	Penalty
LR 0.10	37.37	38.43
LR 0.05	78.63	71.85

Table 4.6: Mean computation time network 3

The following graphs illustrate how the approaches behave with respect to time per iteration given two different learning rates. The graphs confirm the previous results in Table 4.6. A lower LR result in a lower

computation time per iteration in favor of the *penalty* approach. Higher LR shows a lower computation time per iteration in favor of the *accept-reject* approach.

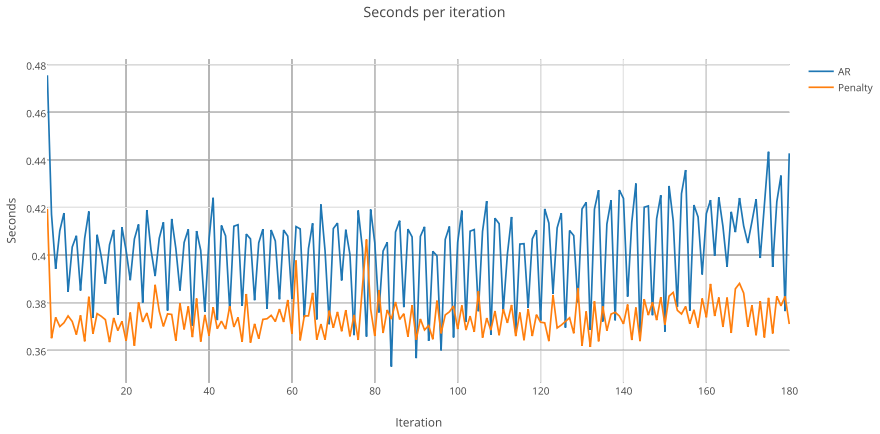

Figure 4.5: Seconds per iteration LR 0.05.

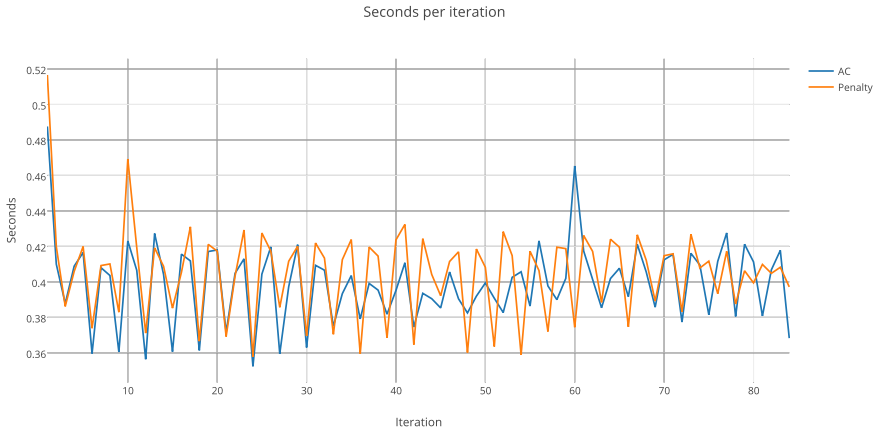

Figure 4.6: Seconds per iteration LR 0.10.

Independent of approach and LR, the suggested solution are the same. Table 4.7 shows the diversity of the various results.

Results from various computations

Suggested solution
'5', '19', '39', '52', '65', '92'
'5', '27', '39', '52', '73', '92'
'5', '19', '40', '60', '65', '84'
'5', '27', '39', '52', '73', '92'
'5', '19', '39', '52', '65', '92'
'5', '27', '39', '60', '65', '84'

Table 4.7: Suggested solution network 3

Increasing the LR dramatically does not provide deviating results, even in a population of 60 computations. Table 4.8 presents an excerpt from the collected results.

Learning rate 0.99 results

Suggested solution
'5', '27', '39', '52', '65', '84'
'5', '19', '39', '52', '73', '92'
'5', '27', '40', '60', '65', '84'
'5', '19', '40', '52', '65', '84'
'5', '19', '40', '60', '65', '92'
'5', '27', '39', '52', '73', '92'

Table 4.8: Incrased learning rate results

Even though the suggested solutions are different from computation to computation, any of the given solutions result in the reduced attack graph in Figure 4.7. The number of privileges an attacker can gain has been reduced from 28 to 7.

Figure 4.7: Reduced attack graph network 3.

Chapter 5

Analysis

The results prove that a network can be hardened with simple steps. All experiments show that the initial attack graphs are reduced dramatically after running the presented tool. The results also show that the given constraints also are maintained.

Experiments on network 1 and 2 result in different suggested solutions, as for instance in Table 4.8. We can observe that all results suggest fixing/patching configuration ID 5 in the first column. However, the other columns show some variations. In the second column, for instance, we see that various results differ between configuration 19 and 27. What is important to realize is that deactivating either of these configurations lead to the same result. Configuration ID 19 and 27 are both AND relations to the same exploit. Deactivating either of them leads to a non-exploitable vulnerability. As stated in Chapter 3, we assume in this thesis that all configurations have the same cost. This tool does therefore not care which configuration should be deactivated as long as the the results provide equal security.

The new probability vector is a weighted combination of the probability vector at the previous time instant and the elite sample. Given a low learning rate, we will put more weight on the previous probability vector and less on the elite sample. The "degree of freedom" of our solution is reduced and is bounded to the previous probability vector.

We can observe in terms of iterations that doubling the learning rate approximately halves the number of iterations needed for all probability vectors to converge. In Chapter 4.3.1 and 4.3.1, we can observe that the methods *acceptance-rejection* and *penalty* have little variation in number of iterations when solving the same task. What also is worth noting is that the *acceptance-rejection* method generates a noticeable greater amount of samples during a computation as illustrated in Figure 4.4. Figure 4.3 tells us that the method generates an enormous amount of samples the first iterations compared with the *penalty* method, and the number decreases exponentially with the number of iterations. In the given

network environments, the time of generating more samples is negligible when comparing the two methods. However, using the *acceptance-rejection* method with a very low threshold and in more complex environments might result in notably increased computation time. What also is important is that a very low threshold can result in a solution which will not harden the network significantly. The user of the tool will need to have a realistic approximation when determining the threshold value, because one cannot necessarily expect the tool to provide a good solution with only being able to suggest just a few fixes/patches in a large network with numerous constraints.

Observing the number of iterations instead of time removes the factor that inefficient code entails. This research has nevertheless measured effectiveness with respect to time. From observing the results, the time taken to compute a solution also doubles when halving the learning rate. However, the effectiveness of two methods prove themselves when changing the learning rate. Figure 4.5 proves that the *penalty* method spends less time per iteration than the *acceptance-rejection* method when having a low learning rate of 0.05. The tables are turned when doubling the learning rate (0.10), then the *acceptance-rejection* method proves itself more effective per iteration in a computation as Figure 4.6 proves.

When increasing the learning rate dramatically, even up to 0.99, the results remain unchanged and the computation time gets reduced enormously. Even though the results in our experiments showed the most optimal solution with the given constraints, increasing the learning rate decreases the reliability of the solution considerably. To consider the results reliable, the learning rate should not exceed 0.1, and lower rates are even more reliable.

Problem solving has been executed on three differently sized networks with different constraints. Based on our experimental results, we can finally state that the suggested tool is scalable, reliable and proves an effective method for constrained optimization with respect to initial configuration-based network hardening. Both methods, *acceptance-rejection* and *penalty* have proven themselves effective to handle constraints for this type of problem solving. For the most reliable results, *penalty* method has proved to be a more effective method. This method is also more generally applicable and is also easy to implement.

Chapter 6

Discussion and Conclusion

In order to ensure full security, a network and system administrator has to take actions at the expense of usability. In some cases, a network can only be regarded safe from potential attackers from the Internet when disconnected from the Internet itself. Doing so is normally not an option for any enterprise or organisation as they rely on providing services to the public. This thesis has found a solution which is able to balance network security and usability.

Vulnerability scanners provide useful information about available vulnerabilities in a network. However, they do not take into account that advanced attackers perform multi-stage attacks where one vulnerability after the other is exploited in order to gain several privileges in a network. Additionally, vulnerability scanners can provide extensive vulnerability reports. The network and system administrator does not know where to start. Attack graphs can show how one vulnerability can lead to another through a relational graph where especially the PRE and POST-conditions of the vulnerabilities are detailed. MulVAL is an example of such a attack graphing tool.

MulVAL provides a foundation for hardening a network using constrained optimization with respect to initial configurations. This research has found that learning automata techniques are convenient for such an optimization. More specifically, Population-Based Incremental Learning, an algorithm suitable for solving combinatorial problems, can be utilized for network hardening optimization. Moreover, this thesis has found that two different methods can be implemented into the PBIL algorithm to ensure that the optimization can be constrained, namely *acceptance-rejection* and *penalty* method. The constraints represent network usability and work load (number of fixes and patches) decided by the network and system administrator.

The developed tool has demonstrated that PBIL is, in fact, convenient for such an optimization. The experiments conducted in this research have shown that complex initial attack graphs can be reduced dramatically by

just a few steps. They also show that this can be done even with extensive constraints. The methods *acceptance-rejection* and *penalty* have both shown themselves effective for implementing constraints into the optimization. However, the *acceptance-rejection* method is prone to increased computation time when having a low threshold in large networks because it results in tremendous amount of rejected samples. Both methods have their merits, but the *penalty* method is more generally applicable.

From a more reflective perspective and invitation for future work, the suggested solution does not take into account that fixing or patching various configurations have different cost. Neither does it take into account that some vulnerabilities lead to heavier impacts than others, nor the likeliness of being exploited. Vulnerability databases and other sources provide helpful information which can be utilized to determine impact, likeliness and cost. Taking these factors into account will provide more intelligent results.

Other types of requirements can also be implemented. The user of the tool would in some cases require a specific attack path to be reduced or require that a particular exploit cannot be exploited. Such an approach could rely on breaking an attack graph down into the various attack paths before computing a solution. Moreover, the results of this research has relied on experiments on non-real networks. Using this approach on a complex real-life running network would give an impression on whether this is useful and effective in practical network and system administration.

In summary, thesis has introduced a method where PBIL can be used to solve network hardening optimization problems with the intention of providing reliable information to quickly correct misconfigurations that may lead to multi-stage attacks in a network. The solution can account for both security and usability requirements through adopting the constraint methods *accept-rejection* and *penalty*. A tool has been developed which can be used in practical network and system administration and relies on initial configurations in order to perform network hardening. Experimental results demonstrate that this approach is effective, scalable and reliable. Lastly, this field of study has enormous potential and this approach to network hardening provides an excellent basis for future works.

Bibliography

Agache, M. & Oommen, B. J. (2002). Generalized pursuit learning schemes: new families of continuous and discretized learning automata. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, *32*(6), 738–749.

Baluja, S. (1994). *Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning*. DTIC Document.

Baluja, S. & Caruana, R. (1995). Removing the genetics from the standard genetic algorithm. In *Machine learning: proceedings of the twelfth international conference* (pp. 38–46).

Baluja, S. & Davies, S. (1997). *Using optimal dependency-trees for combinatorial optimization: learning the structure of the search space.* DTIC Document.

CERT/CC. (2015, April 30). Vulnerability notes database. Retrieved from http://www.kb.cert.org/vuls/

Fogel, D. B. (1994). An introduction to simulated evolutionary optimization. *Neural Networks, IEEE Transactions on*, *5*(1), 3–14.

Folly, K. (2005). Multimachine power system stabilizer design based on a simplified version of genetic algorithms combined with learning. In *Intelligent systems application to power systems, 2005. proceedings of the 13th international conference on* (7–pp). IEEE.

Gent, I. P. & Walsh, T. (1993). Towards an understanding of hill-climbing procedures for sat. In *Aaai* (Vol. 93, pp. 28–33).

Golberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. *Addion wesley*, *1989*.

Greene, J. (1996). Population-based incremental learning as a simple, versatile tool for engineering optimization. In *Proceedings of the first international conf. on ec and applications* (pp. 258–269).

Hiihfeld, M. & Rudolph, G. (1997). Towards theory of population-based incremental learning.

Holm, H., Sommestad, T., Almroth, J., & Persson, M. (2011). A quantitative evaluation of vulnerability scanning. *Information Management & Computer Security*, *19*(4), 231–247.

Homer, J. & Ou, X. (2009). Sat-solving approaches to context-aware enterprise network security management. *Selected Areas in Communications, IEEE Journal on*, *27*(3), 315–322.

Howard, M., Pincus, J., & Wing, J. M. (2005). *Measuring relative attack surfaces*. Springer.

ICS-CERT. (2014, February 20). Ntp reflection attack. Retrieved from https://ics-cert.us-cert.gov/advisories/ICSA-14-051-04

Ingols, K., Lippmann, R., & Piwowarski, K. (2006). Practical attack graph generation for network defense. In *Computer security applications conference, 2006. acsac'06. 22nd annual* (pp. 121–130). IEEE.

Jajodia, S., Noel, S., & O'Berry, B. (2005). Topological analysis of network attack vulnerability. In *Managing cyber threats* (pp. 247–266). Springer.

Lippmann, R. P. & Ingols, K. W. (2005). *An annotated review of past papers on attack graphs*. DTIC Document.

Lozano, J. A. (2000). Analyzing the population based incremental learning algorithm by means of discrete dynamical systems. *Complex Systems*, *12*, 465–479.

Manadhata, P., Wing, J., Flynn, M., & McQueen, M. (2006). Measuring the attack surfaces of two ftp daemons. In *Proceedings of the 2nd acm workshop on quality of protection* (pp. 3–10). ACM.

Meybodi, M. R. & Beigy, H. (2002). New learning automata based algorithms for adaptation of backpropagation algorithm parameters. *International Journal of Neural Systems*, *12*(01), 45–67.

Narendra, K. S. & Thathachar, M. A. (2012). *Learning automata: an introduction*. Courier Corporation.

Narendra, K. S. & Thathachar, M. (1974). Learning automata-a survey. *Systems, Man and Cybernetics, IEEE Transactions on*, (4), 323–334.

Noel, S., Jajodia, S., O'Berry, B., & Jacobs, M. (2003). Efficient minimum-cost network hardening via exploit dependency graphs. In *Computer security applications conference, 2003. proceedings. 19th annual* (pp. 86–95). IEEE.

NVD. (2015, April 22). National vulnerability database. Retrieved from https://nvd.nist.gov/

O'Hare, S., Noel, S., & Prole, K. (2008). A graph-theoretic visualization approach to network risk analysis. In *Visualization for computer security* (pp. 60–67). Springer.

Oommen, B. J. & Agache, M. (1999). A comparison of continuous and discretized pursuit learning schemes. In *Systems, man, and cybernetics, 1999. ieee smc'99 conference proceedings. 1999 ieee international conference on* (Vol. 4, pp. 1061–1067). IEEE.

Security Report 2015. (2015). mnemonic AS. Retrieved from http://www.mnemonic.no/Global/PDF/mnemonic_security%20report_2015.pdf

Ou, X., Govindavajhala, S., & Appel, A. W. (2005). Mulval: a logic-based network security analyzer. In *Usenix security*.

Pamula, J., Jajodia, S., Ammann, P., & Swarup, V. (2006). A weakest-adversary security metric for network configuration security analysis. In *Proceedings of the 2nd acm workshop on quality of protection* (pp. 31–38). ACM.

Phillips, C. & Swiler, L. P. (1998). A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on new security paradigms* (pp. 71–79). ACM.

Rubinstein, R. Y. & Kroese, D. P. (2004). *The cross-entropy method: a unified approach to combinatorial optimization, monte-carlo simulation and machine learning*. Springer Science & Business Media.

Seredyński, F. (1998). Distributed scheduling using simple learning machines. *European Journal of Operational Research*, *107*(2), 401–413.

Sheyner, O. & Wing, J. (2004). Tools for generating and analyzing attack graphs. In *Formal methods for components and objects* (pp. 344–371). Springer.

Unsal, C., Kachroo, P., & Bay, J. S. (1997). Simulation study of multiple intelligent vehicle control using stochastic learning automata. *Transactions of the Society for Computer Simulation*, *14*(4), 193–210.

Wang, L., Albanese, M., & Jajodia, S. (2014). *Network hardening - an automated approach to improving network security*. Springer Briefs in Computer Science. Springer. doi:10.1007/978-3-319-04612-9

Wang, L., Singhal, A., & Jajodia, S. (2007). Toward measuring network security using attack graphs. In *Proceedings of the 2007 acm workshop on quality of protection* (pp. 49–54). ACM.

Yang, S. & Yao, X. (2005). Experimental study on population-based incremental learning algorithms for dynamic optimization problems. *Soft Computing*, *9*(11), 815–834.

Yi, S., Peng, Y., Xiong, Q., Wang, T., Dai, Z., Gao, H., … Xu, L. (2013). Overview on attack graph generation and visualization technology. In *Anti-counterfeiting, security and identification (asid), 2013 ieee international conference on* (pp. 1–6). IEEE.

Appendices

Note that some of the attack graphs might not be readable on paper, but only while enlarging the page in a PDF-reader.

Appendix A

Network 1 attack graph

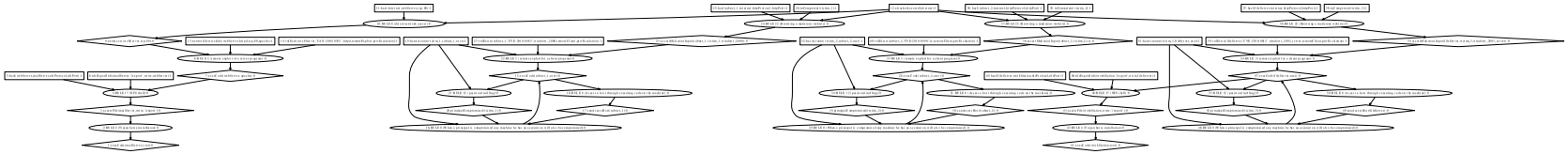

Figure A.1: Initial attack graph network 1

Appendix B

Network 2 attack graph

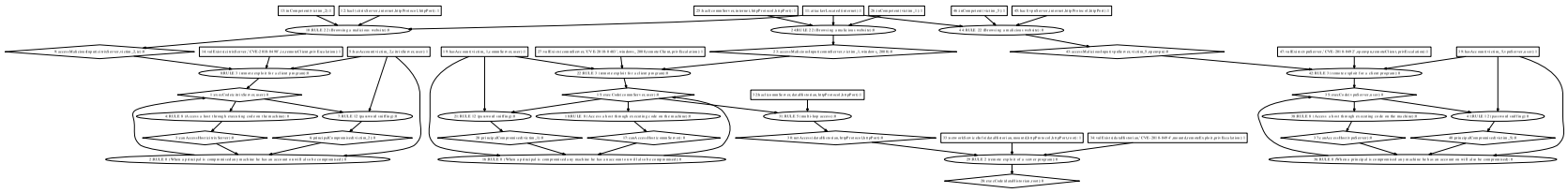

Figure B.1: Initial attack graph network 2

Appendix C

Network 3 attack graph

Figure C.1: Initial attack graph network 3